Progetto N°4 - Calcolo Scientifico

Nodari Alessandro & Proserpio Lorenzo

June 2021

Problema 1

Funzione *segment_ image*

La funzione *segment_ image* prende in ingresso due parametri:

- un'immagine *img* in formato *.PNG*;
- un intero k che rappresenta il numero di *clusters* che vogliamo.

Le immagini con tale formato vengono importate come tensore di *uint8* in MATLAB. Le dimensioni del tensore sono date da: altezza per larghezza (in *pixels*) e numero c di canali di colore. Il valore contenuto in ogni cella, fissato il canale, è pari all'intensità del colore preso in esame sul *pixel* che si trova in quella posizione. Dopodichè converte i valori di ogni cella in *float* a singola precisione (usando il comando *single*) e li normalizza in base all'intensità di colore massima considerata. Poi viene creato tramite *reshape* una matrice I in cui è contenuta una copia dell'immagine; sostanzialmente ogni riga del tipo $1 \times 1 \times c$ del tensore diventa una riga della matrice I. A questo punto si applica l'algoritmo di clustering *k-means* (che verrà discusso nella prossima sottosezione) con la funzione *kmeans*. Il risultato della funzione viene salvato in un vettore *pixels* che contiene il riferimento (per ogni pixel dell'immagine) a quale cluster appartiene contenuto nella matrice *colors* (di tipo $k \times c$) che rappresenta i centroidi. A questo punto viene dichiarata un'altra variabile *I_ qnt* con le stesse dimensioni di I in cui, procedendo per canali di colore, le colonne vengono riempite con i colori a cui si riferisce il vettore *pixels* della matrice *colors*. Dunque *I_ qnt* contiene la nostra immagine quantizzata a cui manca solo di essere portata in dimensioni, scala di intensità del colore e formato delle celle giusto. Per fare cio' non facciamo altro che operazioni inverse a quelle iniziali: un *reshape* per rimetterla sottoforma di tensore, un riscalamento per riportare i valori delle celle su una scala corretta e una conversione da singola precisione a *uint8* (intero senza segno di 1 byte).

Algoritmo di clustering *k-means*

L'algoritmo di clustering non supervisionato *k-means* prende in esame n vettori $\underline{x}$ reali di dimensione d. Lo scopo dell'algoritmo è: fissato $k < n$ determinare la

partizione $\mathcal{P}$ *migliore* di $\mathbb{R}^{d\times k}$, composta da $\{P_1, ..., P_k\}$ sottinsiemi di $\mathbb{R}^d$, per quantizzare ogni osservazione $\underline{x}$. Ovviamente il termine *migliore* va formulato meglio, in questo caso definiamo migliore (per k fissato) la partizione $\mathcal{P}$ rispetto a $\mathcal{S}$ se e solo se:

$$\sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{\underline{x},\underline{y}\in P_i} \|\underline{x} - \underline{y}\|^2 < \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{\underline{x},\underline{y}\in S_i} \|\underline{x} - \underline{y}\|^2$$

dove con $|\cdot|$ intendo la cardinalità dell'insieme. Tradotto concettualmente definiamo migliore quella partizione i cui punti appartenenti allo stesso insieme (anche chiamato *cluster*) sono più vicini, usando un concetto di distanza (nella versione originale quella euclidea), tra loro. Il problema si traduce matematicamente nel risolvere:

$$\arg\min_{\mathcal{P}} \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{\underline{x},\underline{y}\in P_i} \|\underline{x} - \underline{y}\|^2$$

L'algoritmo , dato un insieme iniziale di k *centroidi* $\underline{m}_1^{(0)}, ..., \underline{m}_k^{(0)} \in \mathbb{R}^d$, si articola in soli due step:

1. **Step di assegnazione:** ogni singola osservazione $\underline{x}$ viene assegnata al cluster $P_i^{(t)}$ a cui appartiene il centroide $\underline{m}_i^{(t)}$ che si trova a distanza minima dall'osservazione rispetto a tutti gli altri centroidi.

2. **Step di aggiornamento:** ricalcolo dei centroidi di ogni cluster in base alle osservazioni aggiunte con la formula (componente per componente):

$$m_{i_j}^{(t+1)} = \frac{1}{P_i^{(t)}} \sum_{\underline{x}\in P_i^{(t)}} x_j$$

I due steps vengono ripetuti fino a quando lo step di assegnazione non cambia più l'assegnazione delle osservazioni o fino ad un numero massimo di iterazioni i preimpostato.

Osservazioni del caso in esame e dimensionalità del problema

Nel caso d'esempio stiamo andando a leggere un'immagine *.PNG* 512×512 codificata in RGB (con intensità da 0 a 255). Dunque abbiamo $c = d = 3$ e $n = 512 \cdot 512 = 262144$. Le osservazioni sono date da 262144 vettori di float 1×3 che contengono l'intensità normalizzata (divisa per 255) di ogni canale, ognuno riferito ad un pixel dell'immagine. La dimensionalità del problema è dunque, se andiamo a riaggiornare le componenti su loro stesse al passo precedente, pari a $3 \cdot (n + k)$ (siccome necessito di k centroidi di dimensione 1×3). Se non fisso un numero di iterazioni massimo l'algoritmo ha complessità temporale pari a $\mathcal{O}(n^{dk+1})$, se le fisso invece pari a $\mathcal{O}(nkdi)$. Come ultima osservazione l'algoritmo non trova necessariamente la partizione $\mathcal{P}$ ottima, ma c'era da aspettarselo siccome adotta un chiaro approccio *greedy*.

Scelta dei parametri iniziali

Scelta iniziale dei centroidi

I centroidi $\underline{m}_1^{(0)}, ..., \underline{m}_k^{(0)}$ possono essere scelti in modo diverso. Le strategie più comunemente usate sono:

- scelta in modo casuale tra le n osservazioni, come avviene nell'algoritmo di Lloyd (applicabile con l'opzione *('Start', 'sample')* di *kmeans*);
- scelta in modo casuale con distribuzione uniforme a partire da un *seed* $\underline{x}$ e in base alla distanza da quel punto (che puo' essere o meno un'osservazione);
- determinazione a priori scegliendo manualmente i centroidi;
- scelta applicando l'algoritmo $k-means++$, vengono eseguiti 3 steps: nel primo si sceglie un centroide in modo casuale con distribuzione uniforme, poi a partire da quello e in base alle distanze delle osservazioni dal centroide si modifica il tipo di distribuzione e si sceglie il secondo centroide con distanza dal primo data dalla variabile aleatoria della distribuzione modificata, si ripete il secondo step per tutti gli altri centroidi.

L'ultimo metodo è quello di *default* in MATLAB per la funzione *kmeans*, si è dimostrato che produce risultati migliori dell'algoritmo di Lloyd. Ovviamente ci aspettiamo che scelte diverse dei centroidi iniziali possano portare ad immagini quantizzate diverse. Tale risultato trova riscontro nella pratica, entrambe con $k = 2$ e numero di iterazioni massimo pari a 100:

La figura a sinistra è ottenuta con i centroidi iniziali $\{(0.1, 0.2, 0.3), (0.1, 0.7, 0.8)\}$, quella a destra con $\{(0.465, 0.523, 0.4865), (0.8458, 0.4326, 0.2661)\}$.

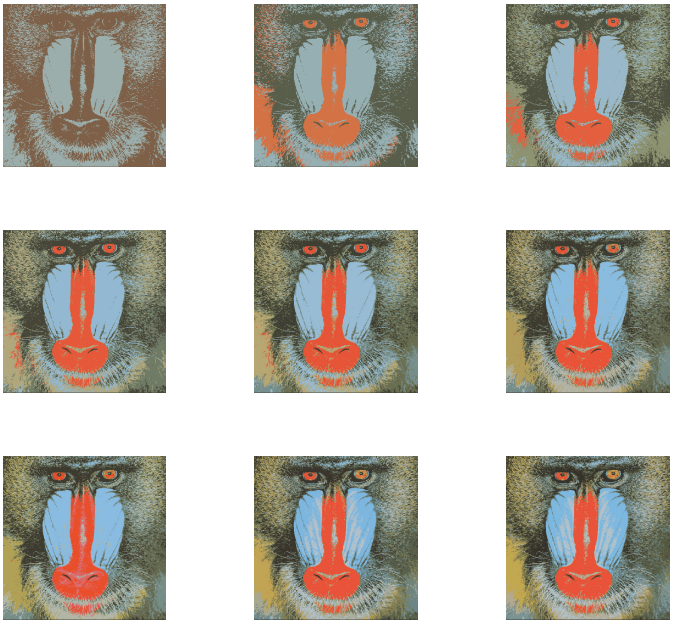
Scelta del numero di iterazioni massimo

Come abbiamo detto nelle sezioni precedenti, l'algoritmo si arresta o quando lo step di assegnazione non modifica più il cluster di appartenenza di ciascuna os-

servazione, oppure quando, se impostato, viene superato il numero di iterazioni massimo. Le due figure ottenute nell'esempio precedente non derivavano da una convergenza dell'algoritmo, ma dal raggiungimento delle iterazioni massime. In linea teorica dovremmo aspettarci, una volta fissato k e i centroidi iniziali di avere un risultato migliore all'aumentare del numero di iterazioni, fino a che non converga definitivamente. Ovviamente la complessità del nostro algoritmo crescerà e ci metterà più tempo all'aumentare del numero di iterazioni. Nella pratica, tenendo $k = 2$ e i centroidi iniziali della figura di sinistra precedentemente presentata, l'algoritmo né converge, anche aumentando il numero massimo di iterazioni a 10^{15}, né produce un risultato significativamente migliore (nelle prossime sezioni vedremo perchè tale scelta di k non è molto adatta). Come possiamo vedere:

Scelta del numero k di cluster

L'algoritmo $k-means$ vuole che si sappia in anticipo il numero k di clusters, che indicano la dimensione del sottospazio in cui vogliamo quantizzare l'immagine. Nel nostro caso specifico possiamo vedere che il numero k non è altro il numero di colori che vogliamo avere nell'immagine quantizzata. Infatti ad ogni cluster corrisponde un centroide che non è altro che un colore in RGB. A seguire i risultati per k che varia tra 2 e 10, con numero di iterazioni massimo pari a 1000:



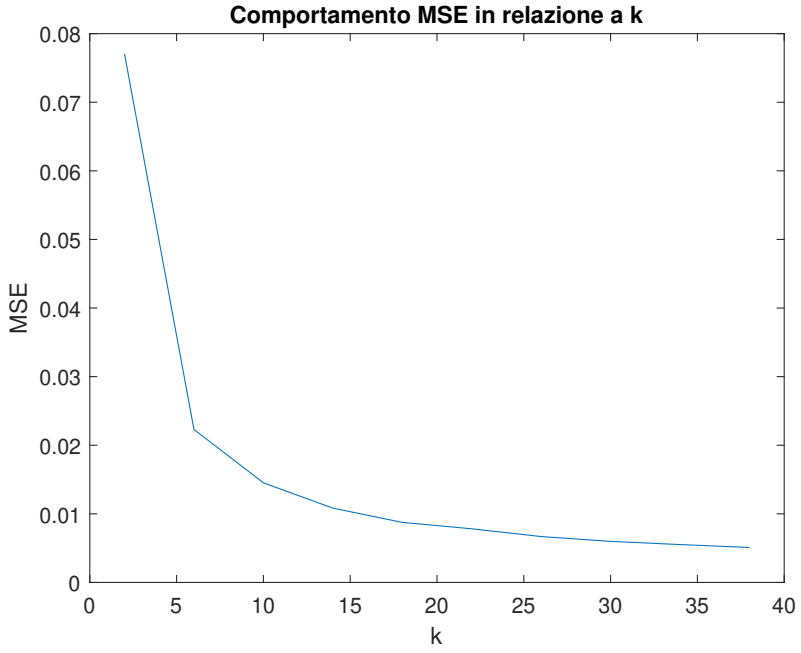
Come possiamo vedere all'aumentare di k otteniamo risultati qualitativamente migliori. Tuttavia, per nessuna delle immagini ottenute l'algoritmo converge, ricordo inoltre che all'aumentare di k aumenta la complessità temporale.

"Stabilità" all'aumentare di k

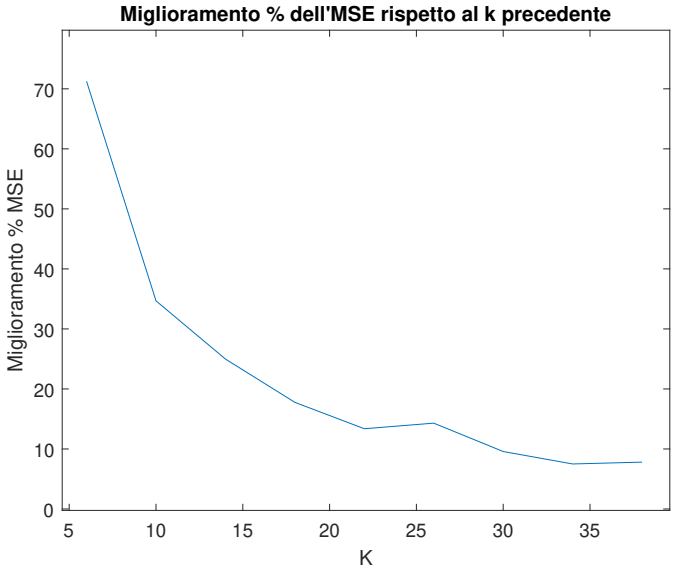
Nella sezione precedente abbiamo visto che al variare della scelta dei centroidi il risultato possa essere differente. All'aumentare di k possiamo osservare che le immagini restano "più stabili", siccome vi è una probabilità maggiore che i colori scelti come centroidi coprino uno "spettro" ragionevole di tonalità, che, anche se vicine, sembrano le stesse ai nostri occhi. Per quantificare tale "stabilità" si potrebbe calcolare la somma delle differenze in termini assoluti pixel per pixel delle immagini ottenute con uno stesso k e centroidi scelti col metodo $k-means++$ e stimare, su un campione ragionevole, la media di tale quantità; poi, ripetere il procedimento per k crescenti. Tuttavia cio' richiede una potenza di calcolo non indifferente, basti pensare che, anche limitando il campione a taglia 10, sarebbero necessarie, per ogni k preso in considerazione circa 10^8 operazioni "elementari".

Individuazione del k "ottimale"

Probabilmente possiamo aspettarci che l'algoritmo non convergerà mai in un numero di iterazioni "ragionevole", visto i dettagli presenti nell'immagine e al fatto che in RGB si possono rappresentare più di 16 milioni di colori differenti, tuttavia possiamo aspettarci che il nostro occhio non sia così fine da distinguerli tutti e di arrivare dunque ad un risultato qualitativamente soddisfacente. Vorremmo a questo punto cercare un metodo matematico per individuare il k che ci dia un risultato qualitativamente e *quantitativamente* migliore o quantomeno un buon compromesso tra risultato e tempo di esecuzione. Per individuare il k ottimale andiamo a valutare l'errore quadratico medio (MSE) per ogni osservazione tra l'immagine originale e quella quantizzata. A tal fine useremo la funzione *segment_ image2.m* e lo script *bestK.m*. Fissiamo anche il numero di iterazioni massime pari a 100 e valutiamo analizziamo i risultati per k che varia da 2 a 38 con step di 4. Otteniamo il seguente grafico:



Vediamo che all'aumentare di k l'MSE diminuisce, ma sempre meno. Per quantificare cio' andiamo a plottare il miglioramento percentuale dell'MSE per ogni valore di k (tolto ovviamente il primo) rispetto al precedente:



Ci si accorge che per $k > 26$ il miglioramento percentuale è inferiore al 12%, mentre la complessità computazionale aumenta linearmente (ricordo che era $\mathcal{O}(nkdi)$). Dunque come valore "ragionevole" prenderemo $k = 26$. Il risultato, a nostro avviso, è qualitativamente soddisfacente, mettiamo l'originale sopra e l'immagine quantizzata sotto di confronto:

Problema 2

Il problema perturbato

Consideriamo il problema P:

$$K \cdot \underline{x} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2^{10}} \end{pmatrix} \underline{x} = \begin{pmatrix} 1 \\ \frac{1}{2^{10}} \end{pmatrix} = \underline{y}$$

la soluzione esatta di tale problema è: $\underline{x} = (1,1)^T$. Siano ora P_1 e P_2 i due problemi perturbati, con termini noti:

$$\underline{y}_{P_1} = (1, 2^{-9})^T \quad \text{e} \quad \underline{y}_{P_2} = (1 + 2^{-10}, 2^{-10})^T$$

le cui rispettive soluzioni sono:

$$\underline{x}_{P_1} = (1,2)^T \quad \text{e} \quad \underline{x}_{P_2} = (1 + 2^{-10}, 1)^T$$

La soluzione esiste sempre ed è unica, siccome la matrice K è a rango pieno (è anche invertibile visto che è quadrata). Vorremmo ora andare ad indagare se il problema P sia *ben condizionato*. Per fare cio', andiamo a calcolare un parametro r che misuri quanto la soluzione reale si discosti da quella perturbata in base all'intensità della perturbazione. Tale r_i è così definito:

$$r_i = \frac{\|\underline{x} - \underline{x}_{P_i}\|_2}{\|p_i\|_2} \quad \text{con } p_i \coloneqq \underline{y} - \underline{y}_{P_i}$$

otteniamo che $r_1 = 1024$ e $r_2 = 1$, eppure abbiamo che le perturbazioni p_1 e p_2 hanno la stessa norma. Da cio' emerge che una piccola perturbazione sulla seconda componente del termine noto porta un grande cambiamento nella soluzione, mentre sul primo no. Cio' era intuibile dalle proprietà spettrali di K. Infatti il numero di condizionamento di K è pari a $\frac{\rho}{\lambda_{min}} = \frac{1}{2^{-10}} = 1024$, dunque K è mal condizionata.

Il problema "regolarizzato"

Analizziamo ora il seguente problema "regolarizzato" P_α:

$$K_\alpha \cdot \underline{x} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2^{10}(1-\alpha)^{10}} \end{pmatrix} \underline{x}$$

con $\alpha \in (0,1)$ e mettendo ancora una volta come termini noti $\underline{y}$, $\underline{y}_{P_1}$ e $\underline{y}_{P_2}$ del punto precedente. Le cui soluzioni sono $\underline{x} = (1, (1-\alpha)^{10})^T$, $\underline{x}_{P_1} = (1, 2 \cdot (1-\alpha)^{10})^T$ e $\underline{x}_{P_2} = (1 + 2^{-10}, (1-\alpha)^{10})^T$. Dunque:

$$r_1 = 2^{10} \cdot (1-\alpha)^{10} \quad \text{e} \quad r_2 = 1$$

cerchiamo l'α tale per cui venga minimizzato r_1 ed è per α più vicino ad 1 possibile, nel nostro caso useremo la precisione macchina e sceglieremo $\alpha = 1 - 10^{-14}$.

Invece r_2 è costante... Il numero di condizionamento di K_α dipende dal valore di α:

$$C(K_\alpha) = \begin{cases} 2^{10}(1-\alpha)^{10} & \alpha \leq 1/2 \\ \dfrac{1}{2^{10}(1-\alpha)^{10}} & \alpha > 1/2 \end{cases}$$

Nel primo caso $C(K_\alpha) \in [1, 2^{10})$, nel secondo $C(K_\alpha) \in (1, +\infty)$. Il minimo di $C(K_\alpha)$ si ha per $\alpha = 1/2$ (cioè $K_\alpha = Id$), per tale valore anche $r_1 = 1$. Con questa "regolarizzazione" abbiamo dunque che il sistema è "stabile" rispetto alle perturbazioni del termine noto in entrambe le componenti. Notiamo che se prendessimo $\alpha = 1 - 10^{-14}$ per rendere r_1 il più piccolo possibile, la matrice sarebbe nuovamente mal condizionata.